LLNL-CONF-666137

Scaling of the Overlink Mesh Interpolation Code on Sequoia

J. M. Grandy

January 13, 2015

Nuclear Explosives Code Development Conference
Los Alamos, NM, United States
October 20, 2014 through October 24, 2014

Scaling of the Overlink Mesh Interpolation Code on Sequoia

Jeffrey M. Grandy
B Division, Lawrence Livermore National Laboratory

Computational Mathematics 30

We demonstrate the scaling behavior of the mesh conversion code Overlink on Sequoia, using MPI and OpenMP threads. We examine the problem size and performance for various combinations of MPI and threads, on one node of Blue Gene Q (the Sequoia architecture), showing that OpenMP threads enable significantly larger numbers of zones per domain. In addition, we demonstrate weak scaling up to one trillion mesh zones on 65536 Sequoia nodes, which is two thirds of the entire machine.

Scaling on one BGQ Node with MPI and Threads

We investigate scaling of the Overlink remap code on one node of Blue Gene Q, examining both memory effects (using maximum problem size as a proxy) and CPU efficiency. Our timing numbers for the single node investigation measure wall time and include reading and writing from disk, and initialization times (which are small on small-scale single node runs). For this study, all of our test meshes contain domains which are perfect Cartesian cubes (Overlink can read and write block structured meshes, but these meshes are generalized to unstructured hexahedral form prior to interpolation from the donor to the target). The target mesh is the same number of zones and zone layout as the donor mesh, but the target mesh is slightly compressed in each direction, so that its physical coordinates lie within the donor mesh.

The remapping is second order, with the interpolation in a donor zone given by

$$f(\vec{x}) = f_0 + (\nabla f)\cdot(\vec{x} - \vec{x}_c)$$

where $\vec{x}_c$ is the centroid of the donor zone, hence ensuring a conservative interpolation that preserves the integral value of f over the volume of the zone, regardless of the value of the gradient, excluding roundoff errors. The gradient of the field is constructed from the field values at neighboring zones, generally requiring MPI communication to fill in a layer of halo zones, for multidomain problems with multiple MPI ranks. Our single compute node tests have a single domain of zones, but the multiple node simulations described in the section below (and most Overlink runs) have multiple domains and utilize halo zones for the gradients. If Overlink is run entirely in shared memory, the halo zones can be obtained directly from neighboring domains in memory, and parallel efficiency may be obtained through OpenMP threads.

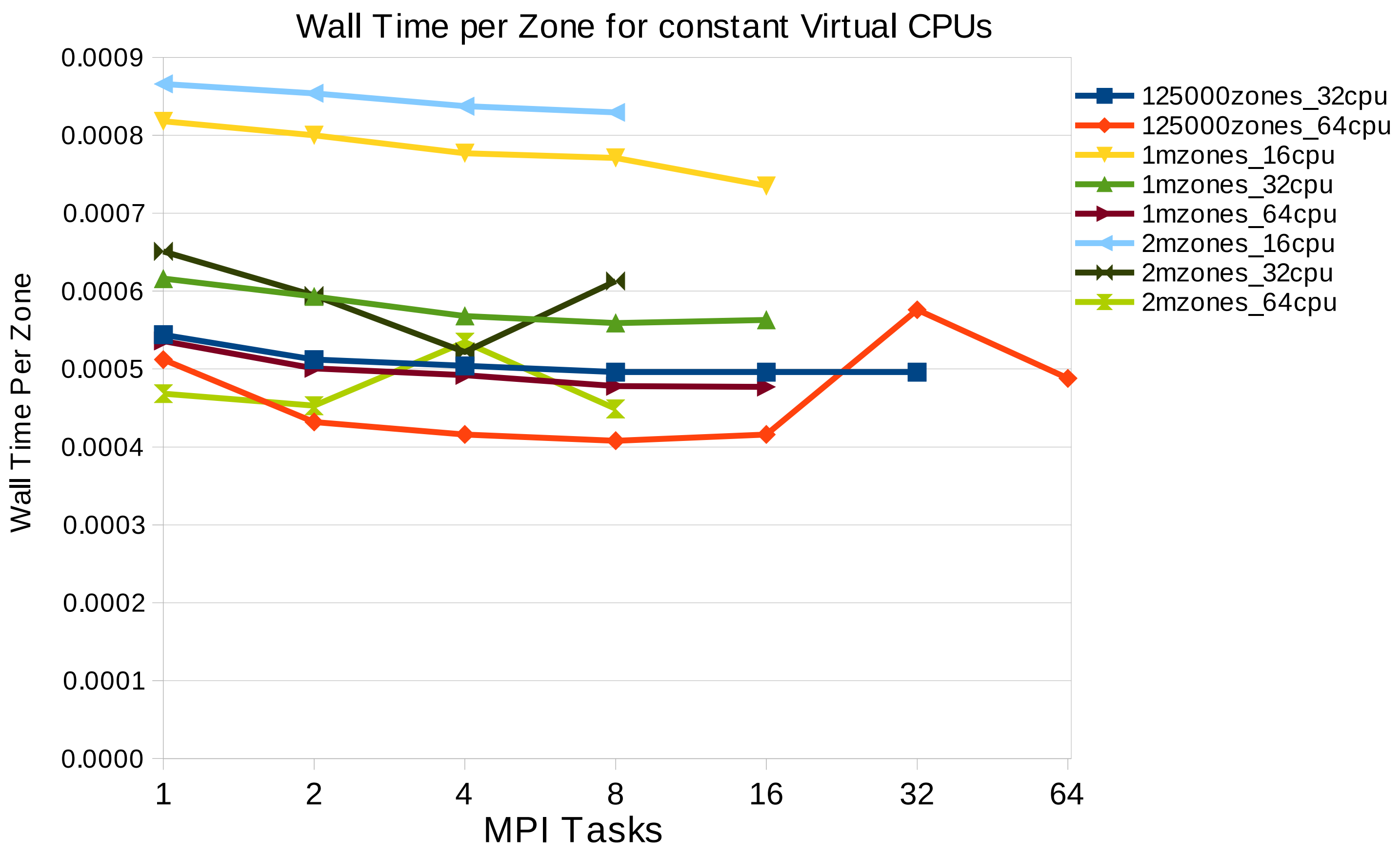
We first calibrate the job launch time by using small-scale runs. The srun command, which is the program used to launch a parallel executable, has an overhead to initialize a run with MPI tasks (even if there is only one MPI task, the code is compiled with the IBM MPI library and executed with the srun launcher). We assume that the measured run time depends on N_z, the total number of zones, the number of MPI tasks M, and the number of OpenMP threads, T, with an additive term for the launch time L.

$$\tilde{t}_w = g(N_z, M, T) + L$$

$$t_w = \tilde{t}_w - L$$

From the calibration we obtain a minimum value of $L \geqslant 4\,sec$ and subtract this from the measured wall time to estimate the wall time spent in the Overlink application, t_w.

We plot the application wall time, per zone, as a function of M, with the colored lines representing a designated total number "virtual cores", $M * T$, and number of zones.



The mesh size of 125000 zones is the largest that would fit on 64 MPI tasks, since the entire domain must be stored on $1/M$ of the node memory to read in the donor and write the result, and the machine architecture strictly partitions the memory among MPI tasks on the node. The plot shows that, particularly with 2 to 16 MPI tasks on the node, the run time varies only slightly when the total number of virtual cores is fixed; this means that utilizing additional CPUs or hardware threads by using OpenMP to generate additional threads is roughly equivalent to using MPI ranks to utilize the CPUs or hardware threads. However, as expected, using more virtual CPUs improves the total run time (note, for example, the runs with 1 m. zones and 16 to 64 virtual CPUs), but, in this range of virtual CPUs, we are already using all 16 physical cores, and the hardware threads, as might be expected, generate less than perfect improvement, as the run with 32 virtual cores takes more than half the time of the run with 16 cores, etc. Also, there is generally only a slight slowdown to run with one MPI rank, indicating that we can take full advantage of the shared memory and obtain reasonable performance.

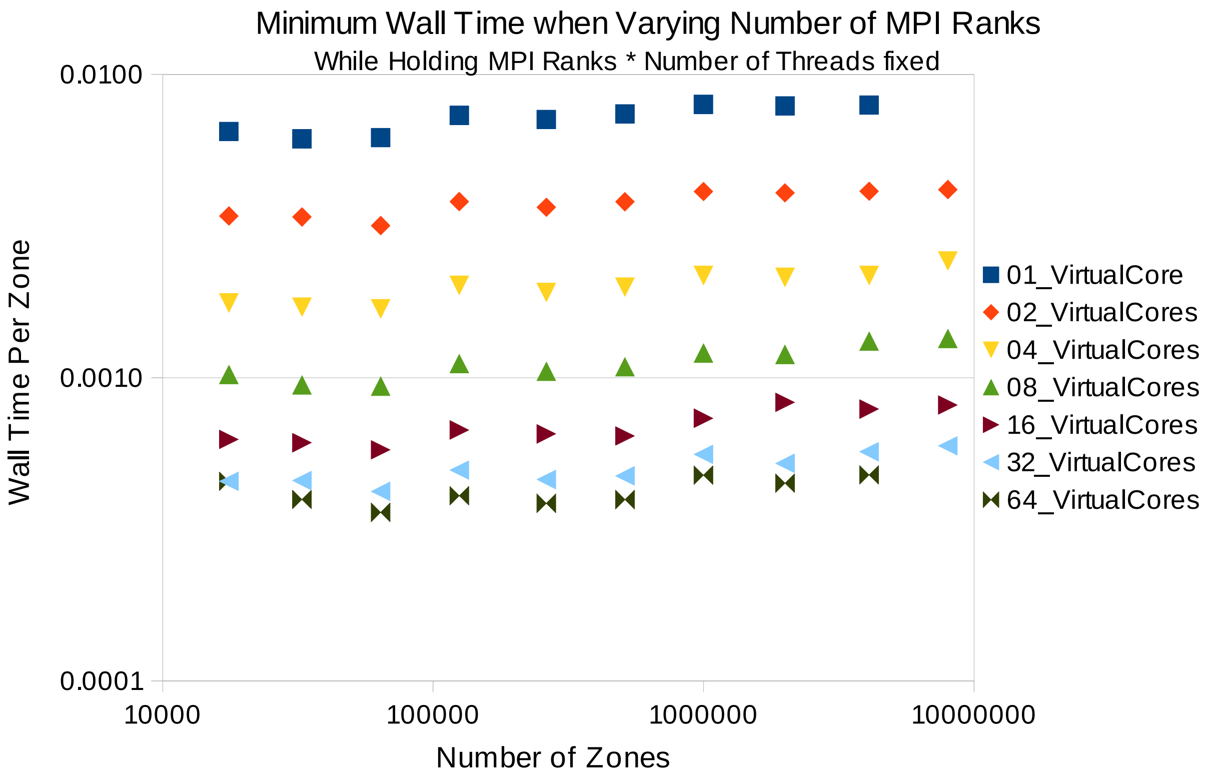
The run time, therefore, depends mostly only on the number of virtual cores,

$$t_w \approx f(N_z, M*T)$$

and perfect scaling would yield

$$t_w / N_z = t_1 / (M*T)$$

where t_1 is the time for one processing core, in serial, to process one donor zone. To examine scaling, we plot the time per zone t_w/N_z as a function of problem size N_z, using one to 64 virtual cores. For each point on the plot, we select the minimum time, for different values of M, for fixed $M*T$. For example, this entails selecting the minimum time on a solid curve in the above graph, as a point on the following graph.

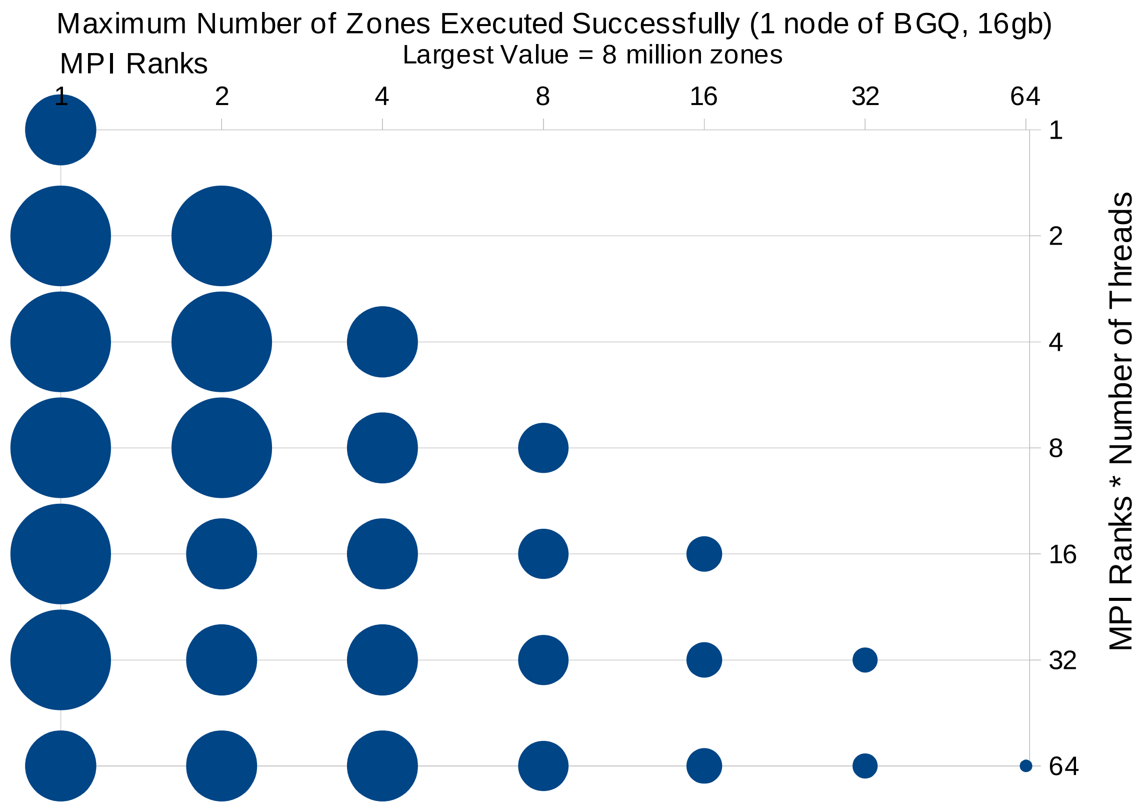


Degradation from scaling for large numbers of virtual cores are partly due to Amdahl's Law, since with $M*T$ of 32 or 64, the serial portion of the run takes roughly half of the run time. For the

8 million zone mesh, the serial run did not complete due to the 16 hour batch system time limit, and the runs with 64 virtual cores did not complete, for any value of M, due to the memory limit, possibly due to temporary objects that are allocated for each thread. As of this writing, we have not performed a detailed study of the memory usage per thread and its effect on maximum problem size. Another degradation is the upward trend of the time per zone as the problem size increases (about 30% for 16 or 32 virtual cores, between 64000 and 8 m. zones). This might be due to the memory cache.

Effect of Shared Memory on Memory Usage

Although the calculation work may be distributed among cores within a node, with a combination MPI tasks and/or OpenMP threads, the number of MPI tasks on the node significantly restricts the maximum domain size that can be used. This is because the memory is strictly partitioned among the MPI tasks, as described above, and a domain must be read and written from a particular task. For a single node of BGQ, the following plot illustrates a lower bound on the maximum problem size (we tested problems in steps of approximately a factor of 2 in number of zones, so it is possible that some of the symbols are "too small" by almost a factor of 2).



Our results show that, while similar performance may be achieved by using MPI or OpenMP parallelism on the node (to get to a particular total parallelism of $M * T$), the usage of OpenMP parallelism (shared memory) provides significant benefit in the flexibility to use a larger domain size. This can reduce total memory usage, for example, by reducing proportionately the total number of halo zones by decreasing the surface area to volume ratio. The ratio of halo zones to real zones decreases (slowly) as $N_z^{-1/3}$ for a cubic mesh with N_z zones, so that for $N_z = 10^3$ zones, one layer of halo zones requires 73% of the number of real zones, while the corresponding number for $N_z = 100^3$ zones is 6.1%. An additional memory benefit, for large meshes distributed across many compute nodes, is in gathering global data. Since Overlink assumes no relationship between the donor and target mesh, any donor domain can potentially intersect any target domain, causing potentially a non-local communications pattern. By contrast, a differencing algorithm that couples only neighboring zones on a single mesh, requires only communication with domains designated as “neighboring”, generally producing a localized communication pattern, each MPI task exchanging data with a small number of other tasks. Therefore, Overlink utilizes arrays enumerated over MPI tasks (or domains), to determine the intersections between domains on the two meshes. An array of k bytes per MPI task, M tasks per node, and N_{comp} compute nodes consumes $k*M*N_{comp}$ bytes per MPI rank (the total number of distributed tasks is $M*N_{comp}$). With M tasks per node, the total memory usage of this array, on the compute node, is

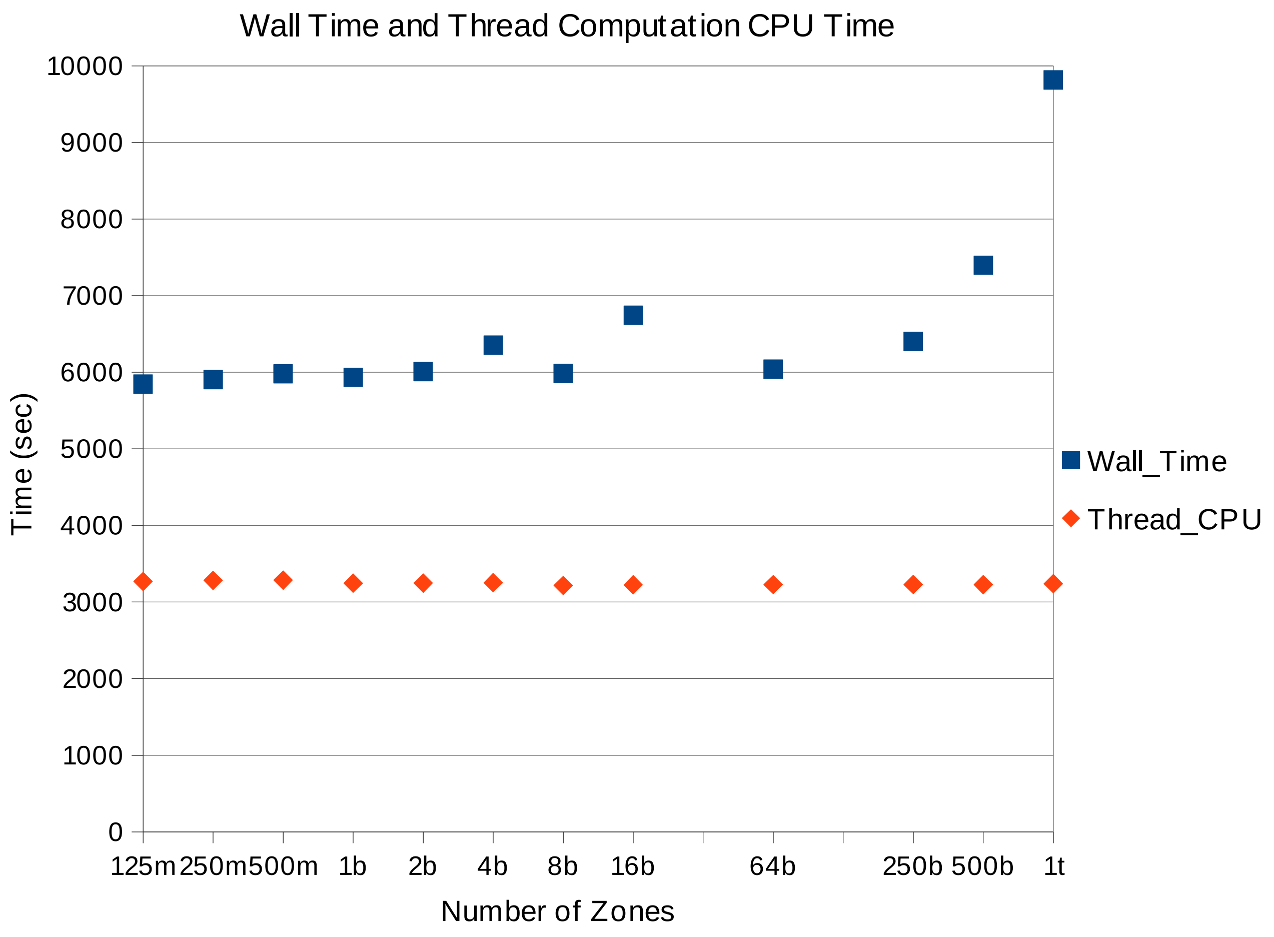
$$n_{bytes} = k*M^2*N_{comp}$$

so that, for example, running with “one task per physical core” (16 tasks per node) uses *256 times* the memory, that would be used with one MPI rank per node. Besides increased halo zones, this produces additional strain on the memory resources of the compute node, for large distributed meshes.

Weak Scaling Study on Sequoia

We have tested the memory usage and performance of Overlink by performing a weak scaling study. We used distributed domain decomposed meshes, and to make more memory available for large numbers of zones, we included only one zone centered field variable in the mesh. We also opted for a modest domain size of 1 m. zones (a cubic mesh 100 zones on a side), in order to provide some flexibility to approximately evenly distribute the same number of zones to each compute node. In this scaling test, we ran Overlink with one MPI rank per node, and 64 OpenMP threads, in order to reduce global memory as described above. Every node participated in reading and writing to the Lustre parallel cluster file system, and we constrained the output to write at most 1000 domain files per subdirectory, each file containing one domain. The target and donor meshes are both the same number of zones and domain decomposition, with a slight shift in the physical coordinates; all donor zones intersected at least one target zone.

During early 2013 we were able to fit 8 domains (8 m. zones) of the donor mesh (and their corresponding target zones) per compute node. However, at that time we found that some arrays of excessive length were allocated within each thread. After fixing this excessive memory usage, we were able to fit at least 16 domains (16 m. zones) on each compute node. For this test, we included only one field variable on the mesh, in order to fit a higher zone count into memory; in the single node study, we used 17 field variables. In the plot below, we show the maximum, over compute nodes, of CPU time (of one of the threads) , for the intersection calculation, and the total wall clock time, for runs of 8 to 65536 compute nodes. The domains were automatically distributed across the compute nodes, so that some nodes had 16 domains and other nodes had 15 domains (for example, the 1 b. zone calculation was 1000 domains on 64 nodes, putting 16 domains on 40 nodes and 15 domains on the other 24).



During our scaling study, conducted from April to June 2013, some of our runs failed due to hangs while writing the result, although improvements in the system were made during the testing process. For the largest run, which required a 65536 node partition (2/3 of the entire Sequoia machine), we needed to run on the weekly “scaling test day” that was scheduled for Sequoia, and thus a failure required a wait of a week or more for another attempt. We explored the possibility of executing the next larger problem (2 t. zones) on the entire Sequoia machine, which would have required at least 21 m. zones (21 domains) per compute node. Although we were able to fit this many domains per node on a small number of compute nodes, we ran out of memory at a smaller scale than the entire machine, perhaps due to the increase of memory usage with problem size described above.

The plot above shows that the interpolation computation phase (the Thread_CPU symbols) is flat, showing embarrassingly parallel scaling (no communication is done during the computation phase), so this is evidence that the parallel work distribution algorithm is working properly. The wall time shows evidence of deviation from scaling, at the level of 500b. zones and 1 t. zones; this might be an effect of the increased time to process arrays over domains, as described earlier. Each compute node does the same amount of disk input/output on all of the runs, since this is a weak scaling test. If the deviation from scaling were to increase, it might have also stopped an attempt to run the next larger problem on the full machine, due to the four hour time limit.

References

1. Grandy, J. 2012. Overlink Users Guide, Lawrence Livermore National Laboratory report LLNL-SM-531271.